BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 7/00

A2

(11) International Publication Number: WO 00/45251

(43) International Publication Date: 3 August 2000 (03.08.00)

(21) International Application Number: PCT/US00/01578

(22) International Filing Date: 24 January 2000 (24.01.00)

(30) Priority Data:
09/240,977 29 January 1999 (29.01.99) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

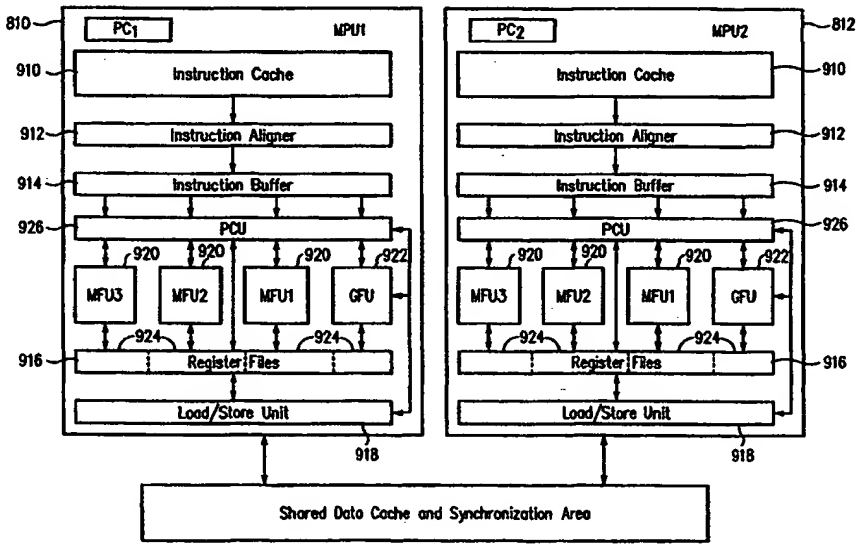
(72) Inventors: SHANKAR, Ravi; 1360 Los Arboles Avenue, Sunnyvale, CA 94087 (US). SUDHARSANAN, Subramania, I.; 4340 Cambridge Way, Union City, CA 94587 (US).

(74) Agents: KOESTNER, Ken, J. et al.; Skjerven, Morrill, MacPherson, Franklin & Friel LLP, Suite 700, 25 Metro Drive, San Jose, CA 95110 (US).

(81) Designated States: JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: PARALLEL FIXED POINT SQUARE ROOT AND RECIPROCAL SQUARE ROOT COMPUTATION UNIT IN A PROCESSOR



(57) Abstract

A parallel fixed–point square root and reciprocal square root computation uses the same coefficient tables as the floating point square root and reciprocal square root computation by converting the fixed–point numbers into a floating–point structure with a leading implicit 1. The value of a number X is stored as two fixed–point numbers. In one embodiment, the fixed–point numbers are converted to the special floating–point structure using a leading zero detector and a shifter. Following the square root computation or the reciprocal square root computation, the floating point result is shifted back into the two–entry fixed–point format. The shift count is determined by the number of leaded zeros detected during the conversion from fixed–point to floating–point format.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

PARALLEL FIXED POINT SQUARE ROOT AND RECIPROCAL SQUARE ROOT COMPUTATION UNIT IN A PROCESSOR

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to computational and calculation functional units of computers, controllers and processors. More specifically, the present invention relates to functional units that execute square root and reciprocal square root operations.

Description of the Related Art

Computer systems have evolved into versatile systems with a vast range of utility including demanding applications such as multimedia, network communications of a large data bandwidth, signal processing, and the like. Accordingly, general-purpose computers are called upon to rapidly handle large volumes of data. Much of the data handling, particularly for video playback, voice recognition, speech process, three-dimensional graphics, and the like, involves computations that must be executed quickly and with a short latency.

One technique for executing computations rapidly while handling the large data volumes is to include multiple computation paths in a processor. Each of the data paths includes hardware for performing computations so that multiple computations may be performed in parallel. However, including multiple computation units greatly increases the size of the integrated circuits implementing the processor. What are needed in a computation functional unit are computation techniques and computation integrated circuits that operate with high speed while consuming only a small amount of integrated circuit area.

Execution time in processors and computers is naturally enhanced through high speed data computations, therefore the computer industry constantly strives to improve the speed efficiency of mathematical function processing execution units. Computational operations are typically performed through iterative processing techniques, look-up of information in

large-capacity tables, or a combination of table accesses and iterative processing. In conventional systems, a mathematical function of one or more variables is executed by using a part of a value relating to a particular variable as an address to retrieve either an initial value of a function or a numeric value used in the computation from a large-capacity table information storage unit. A high-speed computation is executed by operations using the retrieved value. Table look-up techniques advantageously increase the execution speed of computational functional units. However, the increase in speed gained through table accessing is achieved at the expense of a large consumption of integrated circuit area and power.

Two instructions that are highly burdensome and difficult to implement in silicon are a square root instruction and a reciprocal square root operation, typically utilizing many clock cycles and consuming a large integrated circuit area. For example, the square root and the reciprocal square root often have execution times in the range of multiple tens of clock cycles.

For example one technique for computing a square root function or an inverse square root function is to utilize the iterative Newton-Raphson method using a seed value of an approximate value accessed from a lookup table. Hardware for computing the square root or inverse square root includes a multiply/add unit. The iterative technique includes multiple passes through the multiply/add unit. Computation units utilizing the Newton-Raphson method typically take many clock cycles to perform square root and inverse square root operations.

What are needed are a technique for executing square root and reciprocal square root operations, and a computation unit that implements the technique that efficiently execute the operations quickly in a reduced number of clock cycles using a reduced integrated circuit area.

SUMMARY OF THE INVENTION

A parallel fixed-point square root and reciprocal square root computation uses the same coefficient tables as the floating point square root and reciprocal square root computation by converting the fixed-point numbers into a floating-point structure with a

leading implicit 1. The value of a number X is stored as two fixed-point numbers. In one embodiment, the fixed-point numbers are converted to the special floating-point structure using a leading zero detector and a shifter. Following the square root computation or the reciprocal square root computation, the floating point result is shifted back into the two-entry fixed-point format. The shift count is determined by the number of leaded zeros detected during the conversion from fixed-point to floating-point format.

The parallel fixed-point square root and reciprocal square root computation includes several operations. The fixed-point values are normalized into a floating point format include a mantissa and an exponent. The coefficients B and C read are accessed from the storage storing the A, B, and C coefficients used in the floating point computation for both fixed point values. Values D_i are computed by multiplying to obtain the values $B_i x + C_i$ for both of the fixed point numbers. The values D_i are shifted right based on the value of the exponent and placed in fixed-point format.

In accordance with an embodiment of the present invention, a computation unit includes a multiplier and an adder and accesses a storage storing coefficients for computing a piece-wise quadratic approximation. The computation unit further includes a controller controlling operations of the multiplier and adder, and controlling access of the coefficient storage.

In one aspect the storage stores coefficients that are used both for a floating point square root and reciprocal square root computations, and for a parallel fixed-point square root and reciprocal square root computation.

Many advantages are gained by the described method and computation unit. The use of storage devices such as read-only memory (ROM) for storing the coefficients increases the speed of computation. Utilization of the same storage and the same coefficients for both the floating point reciprocal square root computation and the parallel fixed-point reciprocal square root computation efficiently halves the amount of storage allocated to the computations.

BRIEF DESCRIPTION OF THE DRAWINGS

The features of the described embodiments are specifically set forth in the appended claims. However, embodiments of the invention relating to both structure and method of

operation, may best be understood by referring to the following description and accompanying drawings.

FIGUREs 1A and 1B are respectively a schematic block diagram showing an embodiment of a general functional unit and a simplified schematic timing diagram showing timing of a general functional unit pipeline.

FIGURE 2 is a schematic block diagram that illustrates an embodiment of a long-latency pipeline used in the general functional unit.

FIGURE 3 is a graphic shows the format of a single-precision floating point number.

FIGUREs 4A, 4B, and 4C are a plurality of graphs showing exponential functions that describe a technique for performing a single-precision floating-point division operation.

FIGURE 5 is a schematic table that shows a data flow for execution of a floating point reciprocal square root instruction.

FIGURE 6 is a schematic table shows a data flow for execution of the parallel fixed-point reciprocal square root (precsqrt) instruction.

FIGURE 7 is a graphic that shows the format of a fixed point number.

FIGURE 8 is a schematic block diagram illustrating a single integrated circuit chip implementation of a processor in accordance with an embodiment of the present invention.

FIGURE 9 is a schematic block diagram showing the core of the processor.

FIGURE 10 is a schematic block diagram that shows a logical view of the register file and functional units in the processor.

FIGURE 11 is a schematic timing diagram that illustrates timing of the processor pipeline.

The use of the same reference symbols in different drawings indicates similar or identical items.

DESCRIPTION OF THE EMBODIMENT(S)

Referring to **FIGUREs 1A** and **1B** respectively, a schematic block diagram shows an embodiment of a general functional unit **922** (illustrated as part of a processor in **FIGURE 9**) and a simplified schematic timing diagram illustrating timing of general functional unit pipelines **100**. The general functional unit **922** supports instructions that execute in several different pipelines. Instructions include single-cycle ALU operations, four-cycle getir instructions, and five-cycle setir instructions. Long-latency instructions are not fully pipelined. The general functional unit **922** supports six-cycle and 34-cycle long operations and includes a dedicated pipeline for load/store operations.

The general functional unit **922** and a pipeline control unit **926** (also shown more generally in **FIGURE 9**), in combination, include four pipelines, Gpipe1 **150**, Gpipe2 **152**, Gpipe3 **154**, and a load/store pipeline **156**. The load/store pipeline **156** and the Gpipe1 **150** are included in the pipeline control unit **926**. The Gpipe2 **152** and Gpipe3 **154** are located in the general functional unit **922**. The general functional unit **922** includes a controller **160** that supplies control signals for the pipelines Gpipe1 **150**, Gpipe2 **152**, and Gpipe3 **154**. The pipelines include execution stages (En) and annex stages (An).

Referring to **FIGURE 1B**, the general functional unit pipelines **100** include a load pipeline **110**, a 1-cycle pipeline **112**, a 6-cycle pipeline **114**, and a 34-cycle pipeline **116**. Pipeline stages include execution stages (E and En), annex stages (An), trap-handling stages (T), and write-back stages (WB). Stages An and En are prioritized with smaller priority numbers n having a higher priority.

A processor implementing the general functional unit **922** supports precise traps. Precise exceptions are detected by E4/A3 stages of media functional unit and general functional unit operations. One-cycle operations are stages in annex and trap stages (A1, A2, A3, T) until all exceptions in one VLIW group are detected. Traps are generated in the trap-generating stages (T). When the general functional unit **922** detects a trap in a VLIW group, all instructions in the VLIW group are canceled.

When a long-latency operation is in the final execute stage (E6 stage for the 6-cycle pipeline **114** or E34 stage for the 34-cycle pipeline **116**), and a valid instruction is under execution in the A3-stage of the annex, then the long-latency instruction is held in a register, called an A4-stage register, inside the annex and is broadcast to the register file segments **924**

(shown in **FIGURE 9**) only when the VLIW group under execution does not include a one-cycle GFU instruction that is to be broadcast.

Results of long-latency instructions are bypassed to more recently issued GFU and MFU instructions as soon as the results are available. For example, results of a long-latency instruction are bypassed from the E6-stage of a 6-cycle instruction to any GFU and MFU instruction in the decoding (D) stage. If a long-latency instruction is stalled by another instruction in the VLIW group, results of the stalled long-latency instruction are bypassed from the annex (A4) stage to all instructions in the general functional unit **922** and all media functional units **920** (shown in **FIGURE 9**) in the decoding (D) stage.

Data from the T-stage of the pipelines are broadcast to all the register file segments **924**, which latch the data in the writeback (WB) stage before writing the data to the storage cells.

Referring to **FIGURE 2**, a schematic block diagram illustrates an embodiment of a long-latency pipeline **120** used in the general functional unit (GFU) **922**. The long-latency pipeline **120** executes six-cycle instructions. In the illustrative embodiment, the six-cycle instructions include a single-precision floating point division (fdiv) instruction, a single-precision floating point reciprocal square root (frecsqrt) instruction, a fixed-point power computation (ppower) instruction, and a fixed-point reciprocal square root (precsqrt) instruction.

The single-precision floating point division (fdiv) instruction has the form:

fdiv rs1, rs2, rd

where rs1 and rs2 designate a numerator source operand and a denominator source operand, respectively. The rd operand designates a destination register for holding the result.

The single-precision floating point reciprocal square root (frecsqrt) instruction has the form:

frecsqrt rs1, rd

where rs1 designates a source operand and the rd operand identifies the destination register that holds the reciprocal square root result.

The fixed-point power computation (ppower) instruction has the form:

ppower rs1, rs2, rd

where rs1 and rs2 designate source operands and rd identifies a destination register operand. The ppower instruction computes rs1**rs2 for each half of the source registers.

The fixed-point reciprocal square root (precsqrt) instruction has the form:

precsqrt rs1, rd

where rs1 designates a source operand and the rd operand identifies the destination register that holds the reciprocal square root result. The precsqrt instruction computes the reciprocal square root for each half of rs1.

The illustrative long-latency pipeline **120** has eight megacell circuits including a 16-bit normalization megacell **210**, a 24-bit compare megacell **212**, a 16-bit by 16-bit multiplier megacell **214**, an exponent add megacell **216**, a 16-bit barrel shifter megacell **218**, a 25-by-24 multiplier megacell **220**, and a compressor and adder megacell **222**, and a multiplexer and incrementer megacell **224**.

The 16-bit normalization megacell **210** contains a leading zero detector and a shifter that shifts a sixteen bit value according to the status of the leading zero detection. The 16-bit normalization megacell **210** also includes two 4-bit registers that store the shift count values.

The 24-bit compare megacell **212** compares two 24-bit mantissa values. The 24-bit compare megacell **212** generates only equal and less-than signals.

The 16-bit by 16-bit multiplier megacell **214** multiplies two 16-bit values. The actual datapath of the 16-bit by 16-bit multiplier megacell **214** is 18 bit cells wide and includes eight 18-bit rows. The 16-bit by 16-bit multiplier megacell **214** is a radix 4 booth recoded multiplier that generates an output signal in the form of a 32-bit product in binary form. The booth recorders in the 16-bit by 16-bit multiplier megacell **214** are recoded off the binary format in contrast to a carry-save format.

The exponent add megacell **216** subtracts the exponent for a floating point divide operation. The exponent add megacell **216** also performs shifting for execution of a square root operation.

The 16-bit barrel shifter megacell **218** is a 16-bit barrel shifter. The 16-bit barrel shifter megacell **218** is a subset of a 32-bit shifter.

The 25-by-24 multiplier megacell **220** is a 25-bit by 24-bit multiplier. The 25-by-24 multiplier megacell **220** has an actual datapath of 27 bit cells with twelve rows of the 27 bit

cells. The 25-by-24 multiplier megacell **220** is a radix 4 booth recoded multiplier that generates an output signal in the form of a 28-bit product in a carry-save format. The booth recoders are recoded from the carry-save format in contrast to a binary format.

The compressor and adder megacell **222** includes a 4:2 compressor followed by a 28-bit adder. The 28-bit adder uses a kogge-stone algorithm with lings modification.

The multiplexer and incrementer megacell **224** produces two 24-bit products, a sum of two 28-bit numbers in the carry-save format and the increment of the sum. The final multiplexer selects a correct answer based on the sign of the result from the compressor and adder megacell **222**. The adder of the multiplexer and incrementer megacell **224** uses conditional sum adders.

Referring to **FIGURE 3**, a graphic shows the format of a single-precision floating point number **300**. The single-precision floating point format **300** has three fields including one bit for the sign **302**, eight bits for the exponent **304**, and 23 bits for the mantissa **306**. The sign bit **302** equal to zero designates a positive number. The sign bit **302** equal to one designates a negative number. The value of the exponent **304** ranges from 0 to 255. The bias of the exponent **304** is +127. Of the 256 values in the range 0 to 255, only the values of 0 and 255 are reserved for special values. The maximum positive exponent is +127. The minimum negative exponent is –126. The lower order 23 bits designate the mantissa **306**, which is an unsigned fractional number. An implicit value of 1 is included prior to the unsigned fraction. The range of values of the mantissa **306** is from 1.0 to $(2-2^{-23})$. The mantissa range is defined only for normal numbers.

The value of a floating point number is given by the equation, as follows:

$$F = (-1)^{S}1.M(2^{E-127})$$

For the sign bit **302** (S), the mantissa **306** (M), and the exponent **304** (E).

Several special cases are represented differently than the equation in which the floating point number format is otherwise represented as follows:

(1) If the exponent **304** is 255 and the mantissa **306** is zero then the floating point number represents +/- infinity where the sign of infinity is defined by the sign bit **302**.

(2) If the exponent **304** is equal to 255 and M is not equal to zero, then the floating point number is defined as not-a-number (NaN).

(3) If the exponent **304** is equal to zero and the mantissa **306** is equal to zero then the floating point number represents +/- 0. The sign of zero is defined by the sign bit **302**.

(4) If the exponent **304** is equal to zero and the mantissa **306** is not equal to zero, then the floating point number represents a denormal number. The value of the denormal number is given by the equation, as follows:

$$F = (-1)^{S}0.M(2^{E-126})$$

Referring to **FIGUREs 4A, 4B, and 4C**, a plurality of graphs show exponential functions that describe a technique for performing a single-precision floating-point division operation. **FIGURE 4A** is a graph showing an exponential function for computing a square root function $\sqrt{X}$ in an interval i. To determine the coefficients A_i, B_i, and C_i for the reciprocal square root function $\sqrt{X}$, for each interval i, where i is an integer from 0 to 255, 256 equally-spaced points are selected. At each of the 256 points, the value $\sqrt{X}$ is computed in which X is the mantissa. An equation, as follows:

$$\sqrt{X} = \overline{A}_i x_j^2 + \overline{B}_i x_j + \overline{C}_i,$$

is solved for x_j, for a range of integers j from 0 to 255. The values of x_j are the lower-order bits of the mantissa X from x_0 = 0x0 to x_{255} = 0x00000ff, as is shown in **FIGURE 4C**. Solving for x_j produces 256 equations to solve for the coefficients $\overline{A}_i$, $\overline{B}_i$, and $\overline{C}_i$. The coefficient values are obtained by a singular-value decomposition method (see G.H. Golub and C.F. Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, MD. 1983). In one embodiment, a LAPACK software package from the Netlib (www.netlib.org)- Editors Jack Donovan and Eric Grosse is used to compute the coefficients. $\overline{A}_i$, $\overline{B}_i$, and $\overline{C}_i$ are computed for all 256 intervals.

FIGURE 4B is a graph showing an exponential function for computing a reciprocal square root function $1/\sqrt{X}$ in an interval i. To determine the coefficients A_i, B_i, and C_i for the reciprocal square root function $1/\sqrt{X}$, for each interval i, where i is an integer from 0 to

255, 256 equally-spaced points are selected. At each of the 256 points, the value $1/\sqrt{X}$ is computed in which X is the mantissa. An equation, as follows:

$$1/\sqrt{X} = A_i x_j^2 + B_i x_j + C_i,$$

is solved for x_j, for a range of integers j from 0 to 255. The values of x_j are the lower-order bits of the mantissa X from $x_0 = 0x0$ to $x_{255} = 0x00000ff$, as is shown in **FIGURE 4C**. Solving for x_j produces 256 equations to solve for the coefficients A_i, B_i, and C_i using a singular-value decomposition method. A_i, B_i, and C_i are computed for all 256 intervals. The described procedure may also be used to compute a function 1/X.

Referring to **FIGURE 5**, a schematic table shows a data flow for execution of the single-precision floating point reciprocal square root (frecsqrt) instruction. In an illustrative embodiment, the technique for computing the single-precision reciprocal square root of a floating point is the same as the division operation described previously. A piecewise quadratic approximation to $1/\sqrt{X}$ is used. The selected quadratic approximation is described by the equation, as follows:

$$1/\sqrt{X} = Ax^2 + Bx + C.$$

In the illustrative embodiment, a ROM storing 256 words is used for the computation. The 256 word capacity is the same as the ROM storage allocated to the floating point division (fpdiv) operation, but having different coefficients A, B, and C. The result of the reciprocal square root operation is not strictly compliant with the IEEE standard for rounding, supplying a result that possibly differs from results obtained using a technique that complies with the IEEE standard with the result differing by one in some cases.

In a first cycle of the floating point reciprocal square root operation, the source operand x is squared with the result in a binary form using the 16-bit by 16-bit multiplier megacell **214** and values of coefficients A, B, and C are accessed from the ROM using lookup table techniques. In a second cycle, the 16-bit by 16-bit multiplier megacell **214** computes the value Ax^2 in the binary form and the 25-by-24 multiplier megacell **220** computes the value Bx in the carry-save form. In a third cycle, the additions of the Ax^2, Bx and C terms are performed by a 28-bit adder and an 8-bit adder to compute the approximation $1/X^{0.5}$ and (exp2 –127)>>1, respectively.

In a fourth cycle, an additional multiply operation is performed if the exponent is odd. During computation of the square root of a floating point number, the exponent of the result is a simple right shift of the unbiased exponent. However, the simple right shift results in an incorrect result if the unbiased exponent is an odd number. The incorrect result is remedied by multiplying the originally computed result by a constant $B504 (hexadecimal), the value of $\frac{1}{2}^{0.5}$. The 25-by-24 multiplier megacell **220** performs the additional multiply operation with the result in the carry-save format. The 8-bit adder computes the value (exp1-new_exp2).

In a fifth cycle, the result of the floating point reciprocal square root operation is made available. The 28-bit adder converts the quotient Q^1 from carry-save format to binary format. The 8-bit adder decrements the result exponent unless the mantissa **306** is equal to 1.0 and the exponent **304** is even. In a sixth cycle, the result of the floating point reciprocal square root operation is written onto a result bus. The sign of the result is always positive.

Referring to **FIGURE 6**, a schematic table shows a data flow for execution of the parallel fixed-point reciprocal square root (precsqrt) instruction. The technique for computing the reciprocal square root (precsqrt) of two fixed-point numbers is the same as the floating point reciprocal square root method. The technique uses a piecewise quadratic approximation to 1/square root(X). To use the same ROM tables for both the parallel fixed-point and the floating point reciprocal square root operation, the 16-bit fixed point numbers are converted into a floating point-like format with an implicit 1 preceding the fraction of the mantissa. The conversion is performed using a leading zero detector and an appropriate shifter.

In an illustrative embodiment, the multipliers and ROM storages that are used to execute the parallel fixed-point reciprocal square root (precsqrt) are advantageously shared for usage in executing floating-point reciprocal square root (frecsqrt) operations.

The fixed-point number has the standard format S2.13 as shown in **FIGURE 7**, a graphic that shows the format of a fixed-point number **700**. Each 32-bit word includes two fixed point numbers **710** and **720**. The sign of the fixed-point number is always positive. For numbers less than or equal to 0.0625, the result is set to 4.

A ROM for the parallel fixed-point reciprocal square root (precsqrt) instruction is 256 words deep and the stored coefficients A, B, and C are 11 bits, 18 bits, and 28 bits in length, respectively. However, only the B and C coefficients are used in the approximation equation.

The procedure for computing the parallel fixed-point reciprocal square root (precsqrt) instruction is similar to the method for computing the floating point reciprocal square root (frecsqrt) instruction. However, after the computation, the result of the parallel fixed-point reciprocal square root (precsqrt) instruction is shifted back into S2.13 format. The shift count is determined by the number of leading zeros detected during the conversion from the fixed-point format into the floating point format. Like the floating point reciprocal square root operation, and additional multiply is performed if the shift count is odd.

Referring again to **FIGURE 6**, the schematic table shows operations performed in the six cycles of execution of the parallel fixed-point reciprocal square root (precsqrt) instruction. In a first cycle, the 16-bit normalization megacell **210** normalizes the first fixed-point P_1 into a mantissa M_1 and an exponent N_1. In a second cycle, the 16-bit normalization megacell **210** normalizes the second fixed point number P_2 into a mantissa M_2 and an exponent N_2. Also in the second cycle, the ROM is accessed to set the coefficients A_1, B_1, and C_1 for the mantissa M_1.

In a third cycle, the ROM is accessed to set the coefficients A_2, B_2, and C_2 for the mantissa M_2. Also in the third cycle, the 16-bit by 16-bit multiplier megacell **214** computes the value B_1x_1.

During a fourth cycle, the 16-bit by 16-bit multiplier megacell **214** computes the value B_2x_2 and an adder computes the first fixed point number D_1 as the sum of B_1x_1 and C_1.

In a fifth cycle, an additional multiply operation is performed if the exponent is odd. During computation of the square root of the fixed-point number D_1, the exponent of the result is a simple right shift of the unbiased exponent. However, the simple right shift results in an incorrect result if the unbiased exponent is an odd number. The incorrect result is remedied by multiplying the originally computed result by a constant $B504 (hexadecimal), the value of $½^{0.5}$. The 16-bit by 16-bit multiplier megacell **214** performs the additional multiply operation based on the value of the res_exp_1 exponent and formatted into the S2.13 format. Also in the fifth cycle, the adder computes the second fixed point number D_2 as the sum of B_2x_2 and C_2.

In the sixth cycle, the additional multiply operation is performed if the exponent is odd. During computation of the square root of the fixed-point number D_2, the exponent of the result is a simple right shift of the unbiased exponent, resulting in an incorrect result if the

unbiased exponent is an odd number. The incorrect result is corrected for the D_2 number, as for the D_1 number, by multiplying the originally computed result by a constant $B504 (hexadecimal), the value of $\frac{1}{2}^{0.5}$. The 16-bit by 16-bit multiplier megacell **214** performs the additional multiply operation based on the value of the res_exp_2 exponent and formatted into the S2.13 format. The 16-bit by 16-bit multiplier megacell **214** uses the same multiplier and the same adder that are used in floating point operations, thus advantageously eliminating an extra multiplier and adder that would otherwise be required. Also advantageously, the interleaving shown in **FIGURE 6** permits completion of two fixed operations within six cycles without adding extra hardware structures.

Referring to **FIGURE 8**, a schematic block diagram illustrates a single integrated circuit chip implementation of a processor **800** that includes a memory interface **802**, a geometry decompressor **804**, two media processing units **810** and **812**, a shared data cache **806**, and several interface controllers. The interface controllers support an interactive graphics environment with real-time constraints by integrating fundamental components of memory, graphics, and input/output bridge functionality on a single die. The components are mutually linked and closely linked to the processor core with high bandwidth, low-latency communication channels to manage multiple high-bandwidth data streams efficiently and with a low response time. The interface controllers include a an UltraPort Architecture Interconnect (UPA) controller **816** and a peripheral component interconnect (PCI) controller **820**. The illustrative memory interface **802** is a direct Rambus dynamic RAM (DRDRAM) controller. The shared data cache **806** is a dual-ported storage that is shared among the media processing units **810** and **812** with one port allocated to each media processing unit. The data cache **806** is four-way set associative, follows a write-back protocol, and supports hits in the fill buffer (not shown). The data cache **806** allows fast data sharing and eliminates the need for a complex, error-prone cache coherency protocol between the media processing units **810** and **812**.

The UPA controller **816** is a custom interface that attains a suitable balance between high-performance computational and graphic subsystems. The UPA is a cache-coherent, processor-memory interconnect. The UPA attains several advantageous characteristics including a scaleable bandwidth through support of multiple bused interconnects for data and addresses, packets that are switched for improved bus utilization, higher bandwidth, and precise interrupt processing. The UPA performs low latency memory accesses with high throughput paths to memory. The UPA includes a buffered cross-bar memory interface for increased bandwidth and improved scaleability. The UPA supports high-performance

graphics with two-cycle single-word writes on the 64-bit UPA interconnect. The UPA interconnect architecture utilizes point-to-point packet switched messages from a centralized system controller to maintain cache coherence. Packet switching improves bus bandwidth utilization by removing the latencies commonly associated with transaction-based designs.

The PCI controller **820** is used as the primary system I/O interface for connecting standard, high-volume, low-cost peripheral devices, although other standard interfaces may also be used. The PCI bus effectively transfers data among high bandwidth peripherals and low bandwidth peripherals, such as CD-ROM players, DVD players, and digital cameras.

Two media processing units **810** and **812** are included in a single integrated circuit chip to support an execution environment exploiting thread level parallelism in which two independent threads can execute simultaneously. The threads may arise from any sources such as the same application, different applications, the operating system, or the runtime environment. Parallelism is exploited at the thread level since parallelism is rare beyond four, or even two, instructions per cycle in general purpose code. For example, the illustrative processor **800** is an eight-wide machine with eight execution units for executing instructions. A typical "general-purpose" processing code has an instruction level parallelism of about two so that, on average, most (about six) of the eight execution units would be idle at any time. The illustrative processor **800** employs thread level parallelism and operates on two independent threads, possibly attaining twice the performance of a processor having the same resources and clock rate but utilizing traditional non-thread parallelism.

Thread level parallelism is particularly useful for Java™ applications which are bound to have multiple threads of execution. Java™ methods including "suspend", "resume", "sleep", and the like include effective support for threaded program code. In addition, Java™ class libraries are thread-safe to promote parallelism. Furthermore, the thread model of the processor **800** supports a dynamic compiler which runs as a separate thread using one media processing unit **810** while the second media processing unit **812** is used by the current application. In the illustrative system, the compiler applies optimizations based on "on-the-fly" profile feedback information while dynamically modifying the executing code to improve execution on each subsequent run. For example, a "garbage collector" may be executed on a first media processing unit **810**, copying objects or gathering pointer information, while the application is executing on the other media processing unit **812**.

Although the processor **800** shown in **FIGURE 8** includes two processing units on an integrated circuit chip, the architecture is highly scaleable so that one to several closely-coupled processors may be formed in a message-based coherent architecture and resident on the same die to process multiple threads of execution. Thus, in the processor **800**, a limitation on the number of processors formed on a single die thus arises from capacity constraints of integrated circuit technology rather than from architectural constraints relating to the interactions and interconnections between processors.

Referring to **FIGURE 9**, a schematic block diagram shows the core of the processor **800**. The media processing units **810** and **812** each include an instruction cache **910**, an instruction aligner **912**, an instruction buffer **914**, a pipeline control unit **926**, a split register file **916**, a plurality of execution units, and a load/store unit **918**. In the illustrative processor **800**, the media processing units **810** and **812** use a plurality of execution units for executing instructions. The execution units for a media processing unit **810** include three media functional units (MFU) **920** and one general functional unit (GFU) **922**. The media functional units **920** are multiple single-instruction-multiple-datapath (MSIMD) media functional units. Each of the media functional units **920** is capable of processing parallel 16-bit components. Various parallel 16-bit operations supply the single-instruction-multiple-datapath capability for the processor **800** including add, multiply-add, shift, compare, and the like. The media functional units **920** operate in combination as tightly-coupled digital signal processors (DSPs). Each media functional unit **920** has an separate and individual sub-instruction stream, but all three media functional units **920** execute synchronously so that the subinstructions progress lock-step through pipeline stages.

The general functional unit **922** is a RISC processor capable of executing arithmetic logic unit (ALU) operations, loads and stores, branches, and various specialized and esoteric functions such as parallel power operations, reciprocal square root operations, and many others. The general functional unit **922** supports less common parallel operations such as the parallel reciprocal square root instruction.

The illustrative instruction cache **910** has a 16 Kbyte capacity and includes hardware support to maintain coherence, allowing dynamic optimizations through self-modifying code. Software is used to indicate that the instruction storage is being modified when modifications occur. The 16K capacity is suitable for performing graphic loops, other multimedia tasks or processes, and general-purpose Java™ code. Coherency is maintained by hardware that supports write-through, non-allocating caching. Self-modifying code is supported through

explicit use of "store-to-instruction-space" instructions *store2i*. Software uses the *store2i* instruction to maintain coherency with the instruction cache **910** so that the instruction caches **910** do not have to be snooped on every single store operation issued by the media processing unit **810**.

The pipeline control unit **926** is connected between the instruction buffer **914** and the functional units and schedules the transfer of instructions to the functional units. The pipeline control unit **926** also receives status signals from the functional units and the load/store unit **918** and uses the status signals to perform several control functions. The pipeline control unit **926** maintains a scoreboard, generates stalls and bypass controls. The pipeline control unit **926** also generates traps and maintains special registers.

Each media processing unit **810** and **812** includes a split register file **916**, a single logical register file including 128 thirty-two bit registers. The split register file **916** is split into a plurality of register file segments **924** to form a multi-ported structure that is replicated to reduce the integrated circuit die area and to reduce access time. A separate register file segment **924** is allocated to each of the media functional units **920** and the general functional unit **922**. In the illustrative embodiment, each register file segment **924** has 128 32-bit registers. The first 96 registers (0-95) in the register file segment **924** are global registers. All functional units can write to the 96 global registers. The global registers are coherent across all functional units (MFU and GFU) so that any write operation to a global register by any functional unit is broadcast to all register file segments **924**. Registers 96-127 in the register file segments **924** are local registers. Local registers allocated to a functional unit are not accessible or "visible" to other functional units.

The media processing units **810** and **812** are highly structured computation blocks that execute software-scheduled data computation operations with fixed, deterministic and relatively short instruction latencies, operational characteristics yielding simplification in both function and cycle time. The operational characteristics support multiple instruction issue through a pragmatic very large instruction word (VLIW) approach that avoids hardware interlocks to account for software that does not schedule operations properly. Such hardware interlocks are typically complex, error-prone, and create multiple critical paths. A VLIW instruction word always includes one instruction that executes in the general functional unit (GFU) **922** and from zero to three instructions that execute in the media functional units (MFU) **920**. A MFU instruction field within the VLIW instruction word includes an

operation code (opcode) field, three source register (or immediate) fields, and one destination register field.

Instructions are executed in-order in the processor **800** but loads can finish out-of-order with respect to other instructions and with respect to other loads, allowing loads to be moved up in the instruction stream so that data can be streamed from main memory. The execution model eliminates the usage and overhead resources of an instruction window, reservation stations, a re-order buffer, or other blocks for handling instruction ordering. Elimination of the instruction ordering structures and overhead resources is highly advantageous since the eliminated blocks typically consume a large portion of an integrated circuit die. For example, the eliminated blocks consume about 30% of the die area of a Pentium II processor.

To avoid software scheduling errors, the media processing units **810** and **812** are high-performance but simplified with respect to both compilation and execution. The media processing units **810** and **812** are most generally classified as a simple 2-scalar execution engine with full bypassing and hardware interlocks on load operations. The instructions include loads, stores, arithmetic and logic (ALU) instructions, and branch instructions so that scheduling for the processor **800** is essentially equivalent to scheduling for a simple 2-scalar execution engine for each of the two media processing units **810** and **812**.

The processor **800** supports full bypasses between the first two execution units within the media processing unit **810** and **812** and has a scoreboard in the general functional unit **922** for load operations so that the compiler does not need to handle nondeterministic latencies due to cache misses. The processor **800** scoreboards long latency operations that are executed in the general functional unit **922**, for example a reciprocal square-root operation, to simplify scheduling across execution units. The scoreboard (not shown) operates by tracking a record of an instruction packet or group from the time the instruction enters a functional unit until the instruction is finished and the result becomes available. A VLIW instruction packet contains one GFU instruction and from zero to three MFU instructions. The source and destination registers of all instructions in an incoming VLIW instruction packet are checked against the scoreboard. Any true dependencies or output dependencies stall the entire packet until the result is ready. Use of a scoreboarded result as an operand causes instruction issue to stall for a sufficient number of cycles to allow the result to become available. If the referencing instruction that provokes the stall executes on the general functional unit **922** or the first media functional unit **920**, then the stall only endures until the

result is available for intra-unit bypass. For the case of a *load* instruction that hits in the data cache **806**, the stall may last only one cycle. If the referencing instruction is on the second or third media functional units **920**, then the stall endures until the result reaches the writeback stage in the pipeline where the result is bypassed in transmission to the split register file **916**.

The scoreboard automatically manages load delays that occur during a load hit. In an illustrative embodiment, all loads enter the scoreboard to simplify software scheduling and eliminate NOPs in the instruction stream.

The scoreboard is used to manage most interlocks between the general functional unit **922** and the media functional units **920**. All loads and non-pipelined long-latency operations of the general functional unit **922** are scoreboarded. The long-latency operations include division *idiv*, *fdiv* instructions, reciprocal square root *frecsqrt*, *precsqrt* instructions, and power *ppower* instructions. None of the results of the media functional units **920** is scoreboarded. Non-scoreboarded results are available to subsequent operations on the functional unit that produces the results following the latency of the instruction.

The illustrative processor **800** has a rendering rate of over fifty million triangles per second without accounting for operating system overhead. Therefore, data feeding specifications of the processor **800** are far beyond the capabilities of cost-effective memory systems. Sufficient data bandwidth is achieved by rendering of compressed geometry using the geometry decompressor **804**, an on-chip real-time geometry decompression engine. Data geometry is stored in main memory in a compressed format. At render time, the data geometry is fetched and decompressed in real-time on the integrated circuit of the processor **800**. The geometry decompressor **804** advantageously saves memory space and memory transfer bandwidth. The compressed geometry uses an optimized generalized mesh structure that explicitly calls out most shared vertices between triangles, allowing the processor **800** to transform and light most vertices only once. In a typical compressed mesh, the triangle throughput of the transform-and-light stage is increased by a factor of four or more over the throughput for isolated triangles. For example, during processing of triangles, multiple vertices are operated upon in parallel so that the utilization rate of resources is high, achieving effective spatial software pipelining. Thus operations are overlapped in time by operating on several vertices simultaneously, rather than overlapping several loop iterations in time. For other types of applications with high instruction level parallelism, high trip count loops are software-pipelined so that most media functional units **920** are fully utilized.

Referring to **FIGURE 10**, a schematic block diagram shows a logical view of the register file **916** and functional units in the processor **800**. The physical implementation of the core processor **800** is simplified by replicating a single functional unit to form the three media processing units **810**. The media processing units **810** include circuits that execute various arithmetic and logical operations including general-purpose code, graphics code, and video-image-speech (VIS) processing. VIS processing includes video processing, image processing, digital signal processing (DSP) loops, speech processing, and voice recognition algorithms, for example.

A media processing unit **810** includes a 32-bit floating-point multiplier-adder to perform signal transform operations, clipping, facedness operations, sorting, triangle set-up operations, and the like. The media processing unit **810** similarly includes a 16X16-bit integer multiplier-adder for perform operations such as lighting, transform normal lighting, computation and normalization of vertex view vectors, and specular light source operations. The media processing unit **810** supports clipping operations and 1/square root operations for lighting tasks, and reciprocal operations for screen space dividing, clipping, set-up, and the like. For VIS operations, the media processing unit **810** supports 16/32-bit integer add operations, 16X16-bit integer multiplication operations, parallel shifting, and pack, unpack, and merge operations. For general-purpose code, the media processing unit **810** supports 32-bit integer addition and subtraction, and 32-bit shift operations. The media processing unit **810** supports a group load operation for unit stride code, a bit extract operation for alignment and multimedia functionality, a pdist operation for data compression and averaging, and a byte shuffle operation for multimedia functionality.

The media processing unit **810** supports the operations by combining functionality and forming a plurality of media functional units **920** and a general functional unit **922**. The media functional units **920** support a 32-bit floating-point multiply and add operation, a 16X16-bit integer multiplication and addition operation, and a 8/16/32-bit parallel add operation. The media functional units **920** also support a clip operation, a bit extract operation, a pdist operation, and a byte shuffle operation. Other functional units that are in some way incompatible with the media functional unit **920** or consume too much die area for a replicated structure, are included in the general functional unit **922**. The general functional unit **922** therefore includes a load/store unit, a reciprocal unit, a 1/square root unit, a pack, unpack and merge unit, a normal and parallel shifter, and a 32-bit adder.

Computation instructions perform the real work of the processor **800** while load and store instructions may considered mere overhead for supplying and storing computational data to and from the computational functional units. To reduce the number of load and store instructions in proportion to the number of computation instructions, the processor **800** supports group load (*ldg*) and store long (*stl*) instructions. A single load group loads eight consecutive 32-bit words into the split register file **916**. A single store long sends the contents of two 32-bit registers to a next level of memory hierarchy. The group load and store long instructions are used to transfer data among the media processing units **810**, the UPA controller **816**, and the geometry decompressor **804**.

Referring to **FIGURE 11**, a simplified schematic timing diagram illustrates timing of the processor pipeline **1100**. The pipeline **1100** includes nine stages including three initiating stages, a plurality of execution phases, and two terminating stages. The three initiating stages are optimized to include only those operations necessary for decoding instructions so that jump and call instructions, which are pervasive in the Java™ language, execute quickly. Optimization of the initiating stages advantageously facilitates branch prediction since branches, jumps, and calls execute quickly and do not introduce many bubbles.

The first of the initiating stages is a fetch stage **1110** during which the processor **800** fetches instructions from the 16Kbyte two-way set-associative instruction cache **910**. The fetched instructions are aligned in the instruction aligner **912** and forwarded to the instruction buffer **914** in an align stage **1112**, a second stage of the initiating stages. The aligning operation properly positions the instructions for storage in a particular segment of the four register file segments and for execution in an associated functional unit of the three media functional units **920** and one general functional unit **922**. In a third stage, a decoding stage **1114** of the initiating stages, the fetched and aligned VLIW instruction packet is decoded and the scoreboard (not shown) is read and updated in parallel. The four register file segments each holds either floating-point data or integer data.

Following the decoding stage **1114**, the execution stages are performed. The particular stages that are performed within the execution stages vary depending on the particular instruction to be executed. A single execution stage **1122** is performed for critical single-cycle operations **1120** such as, add, logical, compare, and clip instructions. Address-cycle type operations **1130**, such as load instructions, are executed in two execution cycles including an address computation stage **1132** followed by a single-cycle cache access **1134**. General arithmetic operations **1140**, such as floating-point and integer multiply and addition

instructions, are executed in four stages X_1 **1142**, X_2 **1144**, X_3 **1146**, and X_4 **1148**. Extended operations **1150** are long instructions such as floating-point divides, reciprocal square roots, 16-bit fixed-point calculations, 32-bit floating-point calculations, and parallel power instructions, that last for six cycles, but are not pipelined.

The two terminating stages include a trap-handling stage **1160** and a write-back stage **1162** during which result data is written-back to the split register file **916**.

Computational instructions have fundamental importance in defining the architecture and the instruction set of the processor **800**. Computational instructions are only semantically separated into integer and floating-point categories since the categories operate on the same set of registers.

The general functional unit **922** executes a fixed-point power computation instruction *ppower*. The power instruction has the form *ppower r[rs1],r[rs2],r[rd]* and computes *"r[rs1]**r[rs2]"* where each of the sources is operated upon as a pair of independent 16-bit S2.13 format fixed-point quantities. The result is a pair of independent 16-bit S2.13 format fixed-point powers placed in the register *r[rd]*. Zero to any power is defined to give a zero result.

The general functional unit **922** includes functional units that execute a floating-point division *fdiv* instruction, a floating-point reciprocal *frecip* instruction, a floating-point square root *fsqrt* instruction, and a floating-point reciprocal square root *frecsqrt* instruction, each for single-precision numbers. The floating-point division instruction has the form *fdiv rs1,rs2,rd* and computes a single-precision floating-point division *"r[rs1]/r[rs2]"* and delivers the result in *r[rd]*. The floating-point reciprocal instruction has the form *frecip rs1,rd* and executes a single-precision floating-point reciprocal with a latency of eight cycles. The floating-point square root instruction has the form *fsqrt rs1,rd* and executes a single-precision floating-point square root operation. The floating-point reciprocal square root instruction has the form *frecsqrt rs1,rd* and executes a single-precision floating-point reciprocal of the square root operation on the quantity in *r[rs1]* and places the result in *r[rd]*.

The general functional unit **922** also supports a fixed-point parallel reciprocal square root *precsqrt* instruction. The fixed-point reciprocal square root instruction has the form *precsqrt rs1,rd*. *Precsqrt* computes a pair of S2.13 format fixed-point reciprocal square roots of the pair of S2.13 format values on register *r[rs1]*. Results are delivered in register *r[rd]*. The result for a source operand that is less than or equal to zero is undefined.

The general functional unit 922 executes an integer divide *idiv* instruction that computes either *"r[rs1]/r[rs2]"* or *"r[rs1]/sign_ext(imm14)"* and places the result in *r[rd]*.

While the invention has been described with reference to various embodiments, it will be understood that these embodiments are illustrative and that the scope of the invention is not limited to them. Many variations, modifications, additions and improvements of the embodiments described are possible. For example, those skilled in the art will readily implement the steps necessary to provide the structures and methods disclosed herein, and will understand that the process parameters, materials, and dimensions are given by way of example only and can be varied to achieve the desired structure as well as modifications which are within the scope of the invention. Variations and modifications of the embodiments disclosed herein may be made based on the description set forth herein, without departing from the scope and spirit of the invention as set forth in the following claims.

WHAT IS CLAIMED IS:

1. A method of computing a reciprocal square root of a number X in a computing device comprising:

computing a piece-wise quadratic approximation of the number X using an equation of the form:

$1/\sqrt{X} = Ax^2 + Bx + C$, the number X having a mantissa and an exponent, the computing operation including:

accessing the A, B, and C coefficients from a storage;

computing the value Ax^2+Bx+C result, the result having a mantissa and an exponent;

shifting the result exponent right by one bit;

correcting an error that occurs when an asserted bit in the result exponent is discarded by the shifting operation, the correcting operation including multiplying the computed result by a correction constant, wherein the number X includes two fixed point numbers for which the square root computation takes place in parallel using the same coefficient tables as are used for a calculation of a floating point square root.

2. A method of computing a square root of a number X in a computing device comprising:

computing a piece-wise quadratic approximation of the number X having a mantissa and an exponent in a plurality of parallel data paths using an equation of the form:

$\sqrt{X} = Ax^2 + Bx + C$, the computing operation including:

accessing the A, B, and C coefficients from a storage;

squaring the x term of the number X to obtain an x^2 term;

multiplying the x^2 term times the coefficient A to obtain an Ax^2 term;

multiplying the x term times the coefficient B to obtain a Bx term;

summing the Ax^2 term, the Bx term, and the C term to form an approximation result;

shifting the exponent right; and

multiplying the approximation result by a correction constant, wherein:

the number X includes two fixed point numbers for which the square root computation takes place in parallel using the same coefficient tables as are used for a calculation of a floating point square root.

3. A method of computing a reciprocal square root of two fixed-point numbers in a computing device comprising:

converting the two fixed-point numbers into a floating-point number X with a leading implicit 1;

computing a piece-wise quadratic approximation of the floating point number X using an equation of the form:

$1/\sqrt{X} = Ax^2 + Bx + C$, the number X having a mantissa and an exponent, the computing operation including:

accessing the A, B, and C coefficients from a storage;

computing the value Ax^2+Bx+C floating point result, the floating point result having a mantissa and an exponent;

shifting the result exponent right by one bit; and

correcting an error that occurs when an asserted bit in the result exponent is discarded by the shifting operation, the correcting operation including multiplying the computed result by a correction constant; and

shifting the floating point result into the form of two fixed point numbers.

4. A method of computing a square root of two fixed-point numbers in parallel in a computing device comprising:

deriving coefficients A, B, and C for computing a piece-wise quadratic approximation of the floating point number X using an equation of the form:

$1/\sqrt{X} = Ax^2 + Bx + C$,

the coefficients being derived to reduce least mean square error using a least squares approximation of a plurality of equally-spaced points within an interval;

normalizing the fixed-point values into a floating point format including a mantissa and an exponent;

accessing coefficients B and C from the derived coefficients;
computing values $D_i = B_i x + C_i$ for fixed point numbers i=1 and i=2; and
shifting the values D_i right based on the value of the exponent.

5. A method according to either Claim 3 or Claim 4 wherein:
the fixed-point reciprocal square root computation uses a same set of coefficients A, B, and C as a floating point reciprocal square root computation.

6. A method according to any of Claims 3 through 5 further comprising:
converting two fixed-point numbers into the floating-point number X; and
interleaving computing operations so that floating point computation elements are used to perform computations on fixed point numbers in parallel.

7. A method according to any of Claims 3 through 5 further comprising:
converting two fixed-point numbers into the floating-point number X; and
interleaving computing operations so that floating point computation elements are used to perform computations on fixed point numbers in parallel and within six cycles.

8. A method according to Claim 3 wherein:
converting the two fixed-point numbers into a floating-point number X with a leading implicit 1 includes:
detecting the number of leading zeroes in the fixed point number; and
shifting to cancel the leading edge zeroes.

9. An integrated circuit including:
a multiplier;
an adder coupled to the multiplier; and
a control logic coupled to the multiplier and the adder, the control logic performing the method according to any of Claims 3 through 8.

10. An integrated circuit according to Claim 9 wherein:
the multiplier, the adder, and the control logic are alternatively used to perform computations on floating point numbers and fixed point numbers.

11. A processor comprising:
an instruction storage;
a register file coupled to the instruction storage;
a functional unit including:
a multiplier;
an adder coupled to the multiplier; and
a control logic coupled to the multiplier and the adder, the control logic performing the method according to any of Claims 3 through 8.

12. A processor according to Claim 11 wherein:
the multiplier, the adder, and the control logic are alternatively used to perform computations on floating point numbers and fixed point numbers.

13. A computation unit comprising:
a multiplier;
an adder coupled to the multiplier;
a connection to a storage, the storage storing coefficients for computing a piece-wise quadratic approximation; and
a control logic controlling the adder, the multiplier, and access of the storage, the control logic controlling computation of a piece-wise quadratic approximation of the floating point number X using an equation of the form: $1/\sqrt{X} = Ax^2 + Bx + C$, wherein:
the number X includes two fixed point numbers for which the square root computation takes place in parallel using the same coefficient tables as are used for a calculation of a floating point square root.

14. An integrated circuit according to Claim 13 wherein:
the multiplier, the adder, and the storage are alternatively used to perform computations on floating point numbers and fixed point numbers.

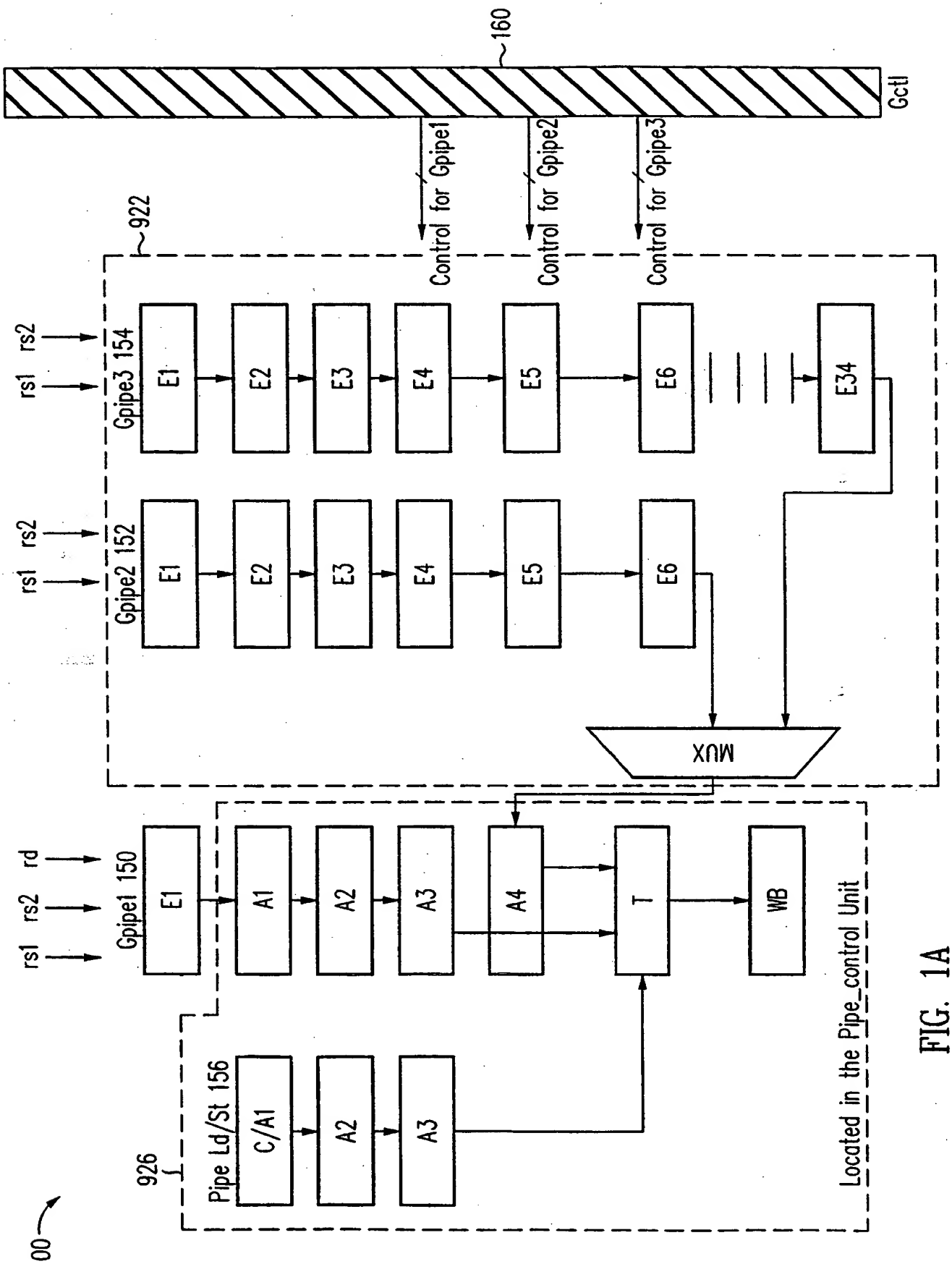


FIG. 1A

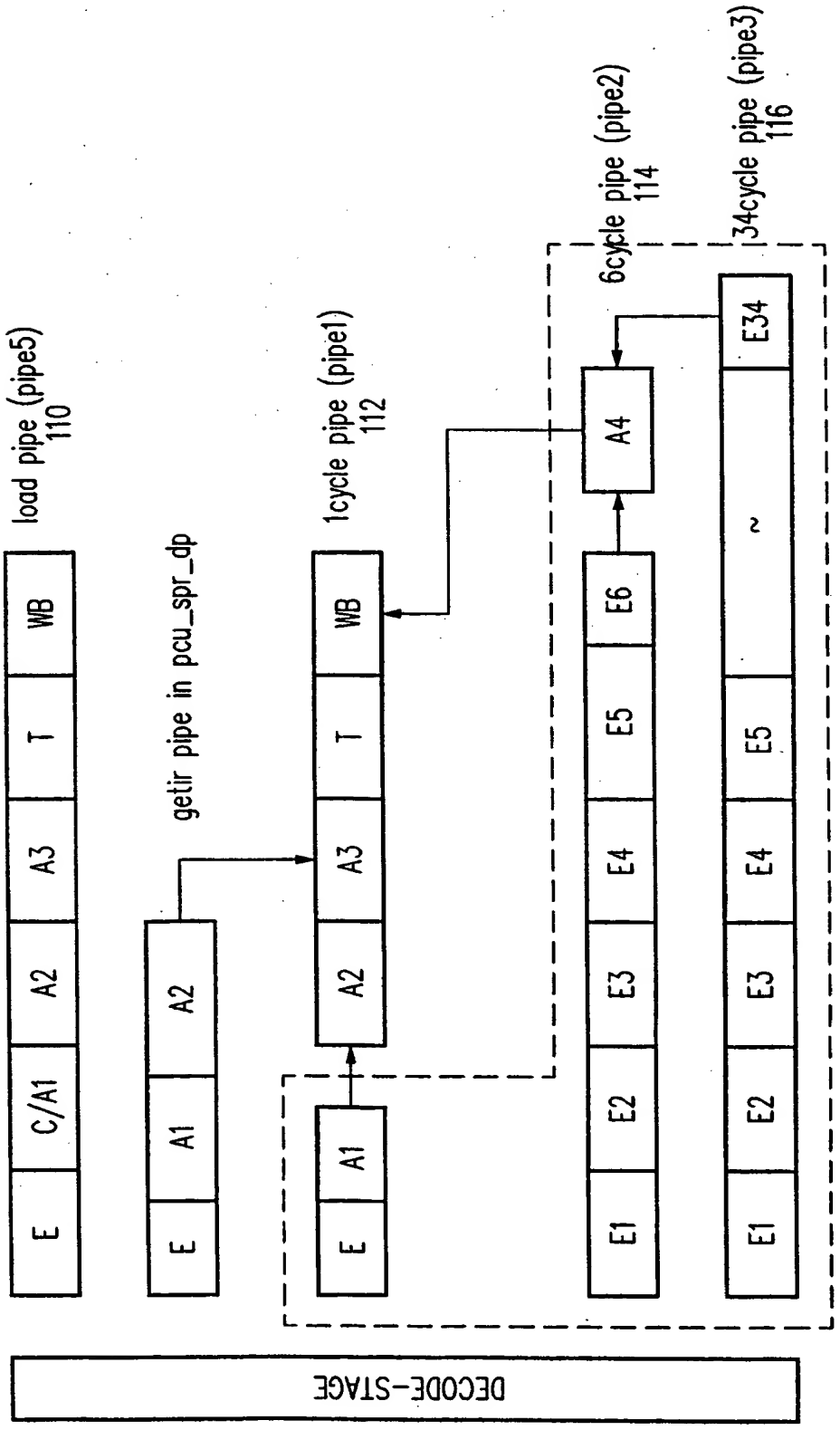


FIG. 1B

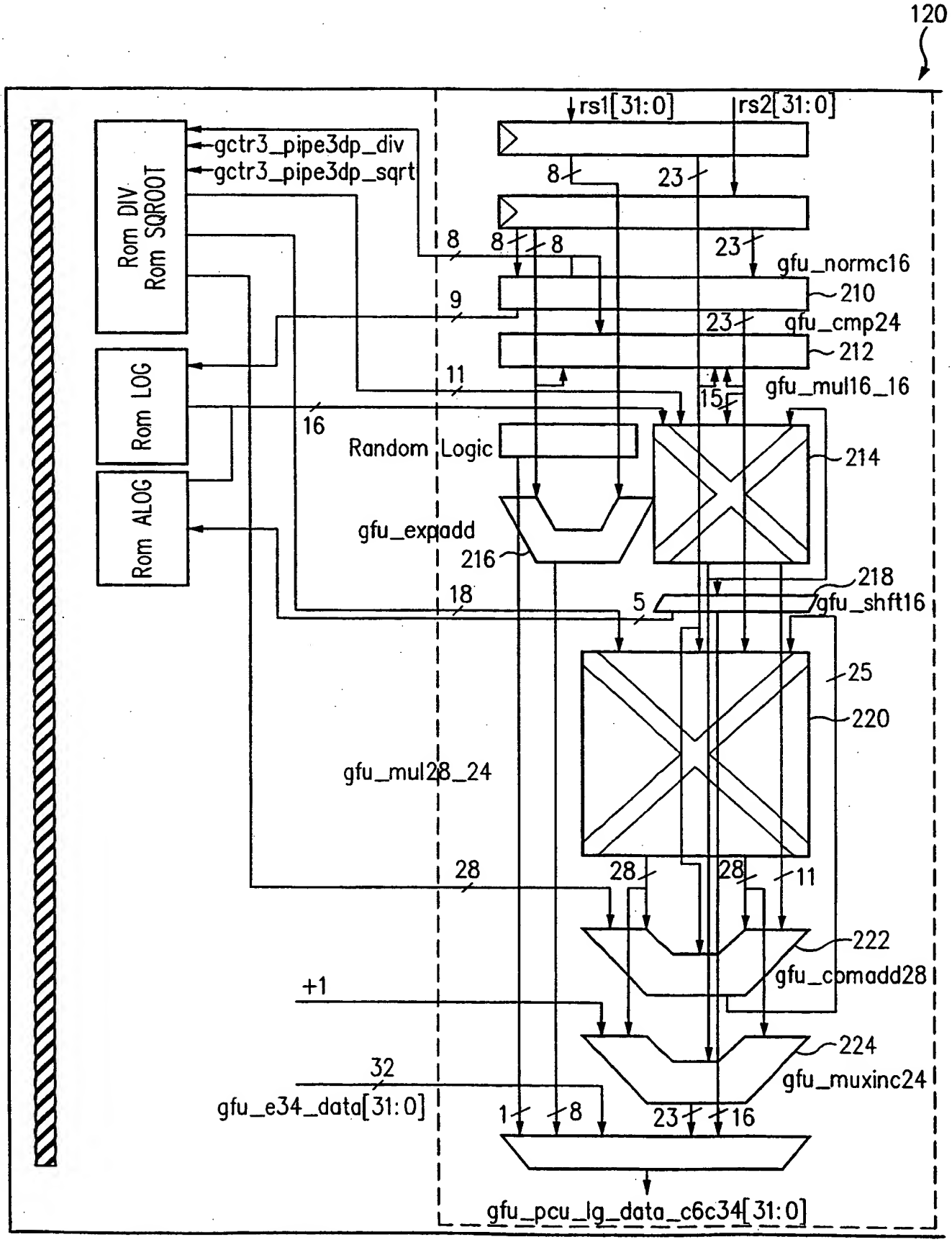


FIG. 2

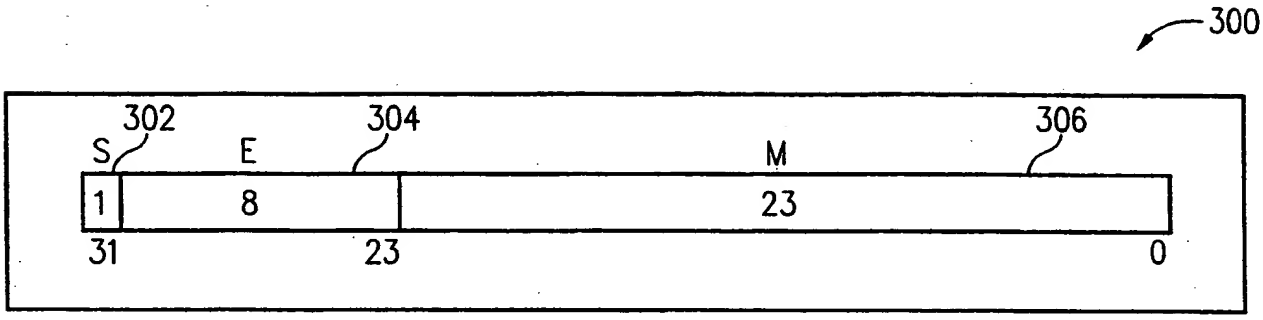


FIG. 3

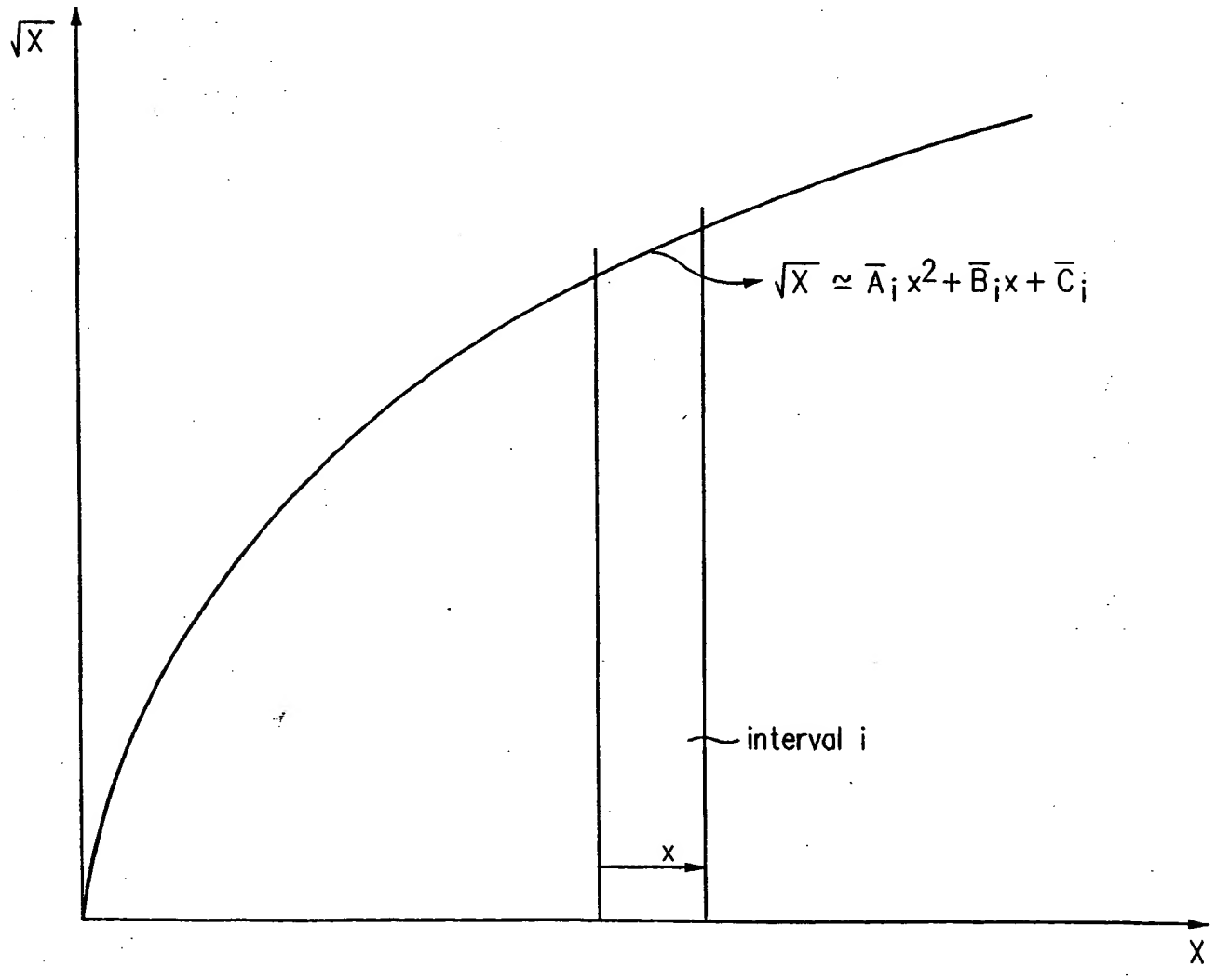


FIG. 4A

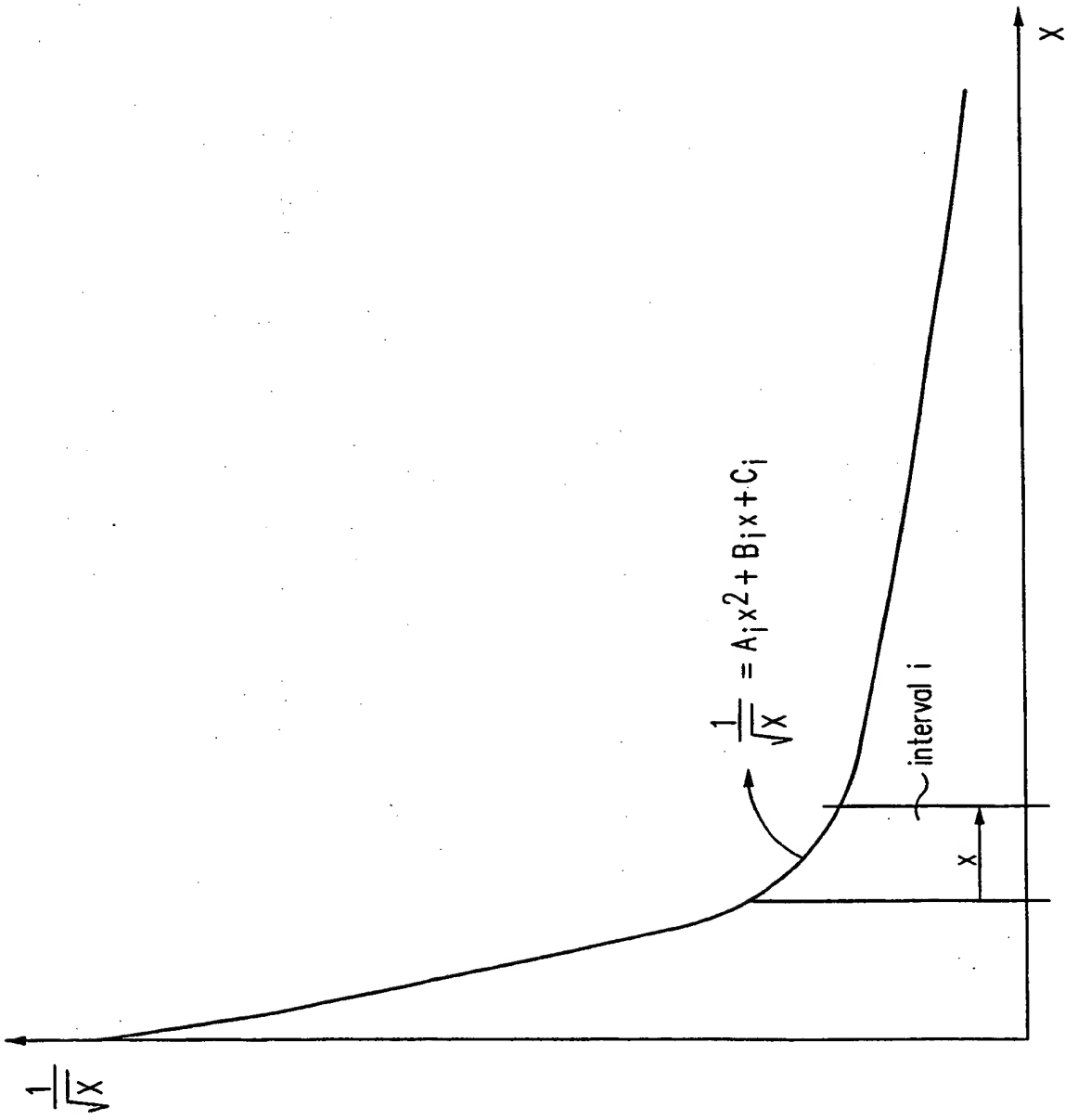


FIG. 4B

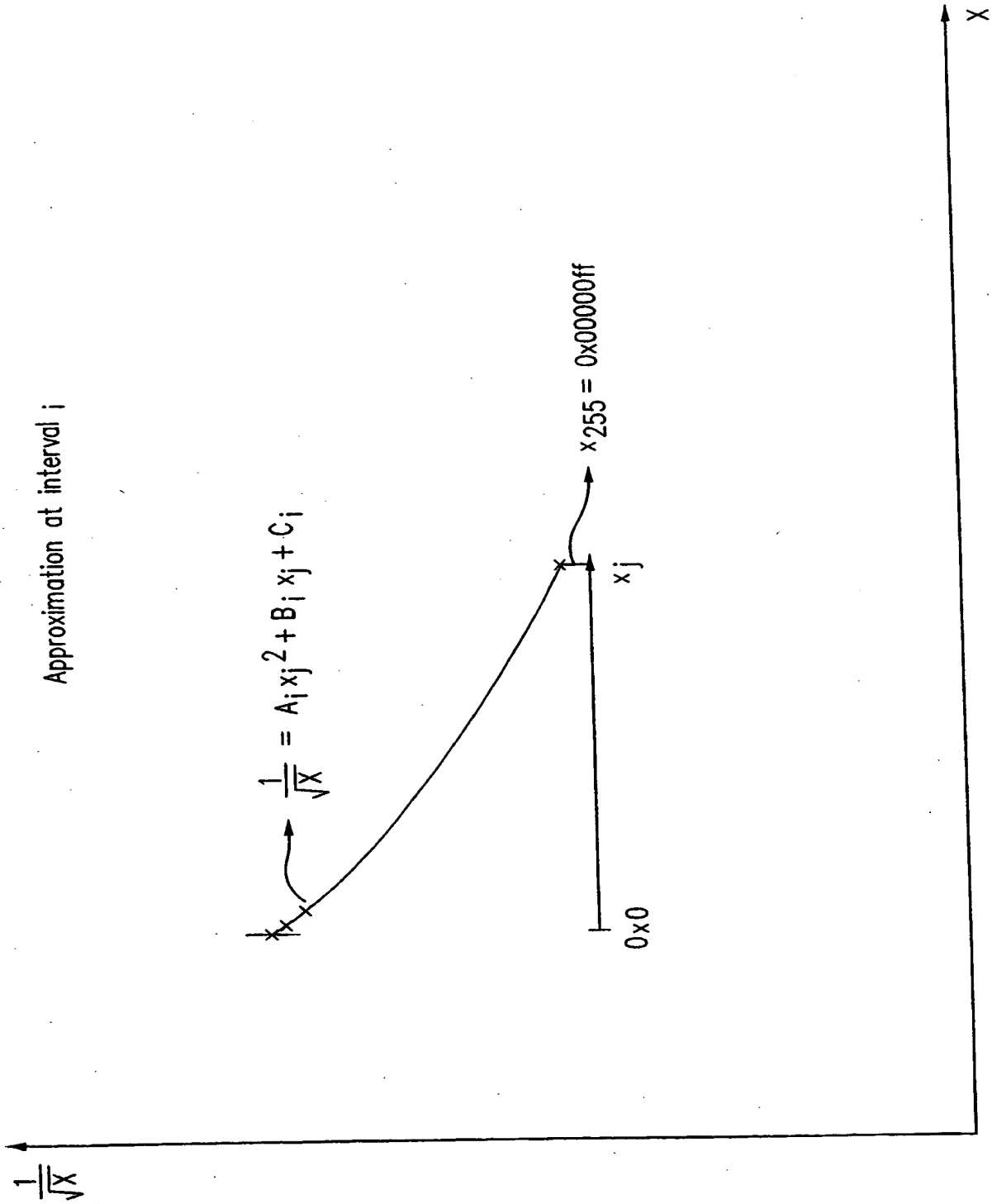


FIG. 4C

Flow of Data for the Floating Point Reciprocal Square Root

Cycle	ROM	16X16 Multiplier	28X24 Multiplier	28 bit Adder	8 bit Adder	Description
1	Look Up A, B & C	x*x(B)				Calculate x^2. Look Up A, B & C
2		Ax^2(B)	Bx(CS)			
3				$1/X^{0.5} = Ax^2+Bx+C$	exp2_un-127	Approx.$1/X^{0.5}$ Calculate (exp2 - 127)>>1.
4			$Q^1=(1/X^{0.5})$*$B504 F3(CS)		res_exp= exp1-new_exp2	Need this additional multiply if the exponent is odd. Calculate (exp1-new_exp2).
5				Convert Q^1 (CS)-> Q^1 (B)	res_exp= res_exp-1	Result available in this cycle. Decrement res_exp unless mant2=1.0 and exp2 is even.
6				Write result onto the result bus.		

FIG. 5

Format of the fixed point number format

Flow of Data for Parallel Fixed Point Reciprocal Square Root

Cycle	Normalizer	ROM	16X16 Multiplier	24X27 Multiplier	28 bit Adder	8 bit Adder	Description
1	Normal-ize P_1 into mant M_1 and expo-nent N_1.						
2	Normal-ize P_2 into mant M_2 and expo-nent N_2.	Look Up A_1, B_1 & C_1 for M_1				exp1_unb=exp1 -7f	Look up B_1 & C_1. Calcu-late unbi-ased exponent1.
3		Look Up A_2, B_2 & C_2 for M_2		B_1x_1		res_exp 1=7f/ 7e-(exp1_u nb>>1)	Compute B_1x_1. Look up B_2 & C_2. Calculate result exp.
4				B_2x_2	$D_1=B_1x_1$ $+C_1$	exp2_un b=exp2 -7f	Compute D_1. Compute B_2x_2.
5			D_1*$B504 Right Shift based on the value of res_exp_1 and fit into S2.13.		$D_2=B_2x_2$ $+C_2$	res_exp 2=7f/ 7e-(exp2_u nb>>1)	Need this additional multiply if the expo-nent is odd. Denormal-ize result back into S2.13. Cal-culate result exp.
6			D_2*$B504 Right Shift based on the value of res_exp_2 and fit into S2.13.				Need this additional multiply if the expo-nent is odd. Denormal-ize result back into S2.13.

FIG. 6

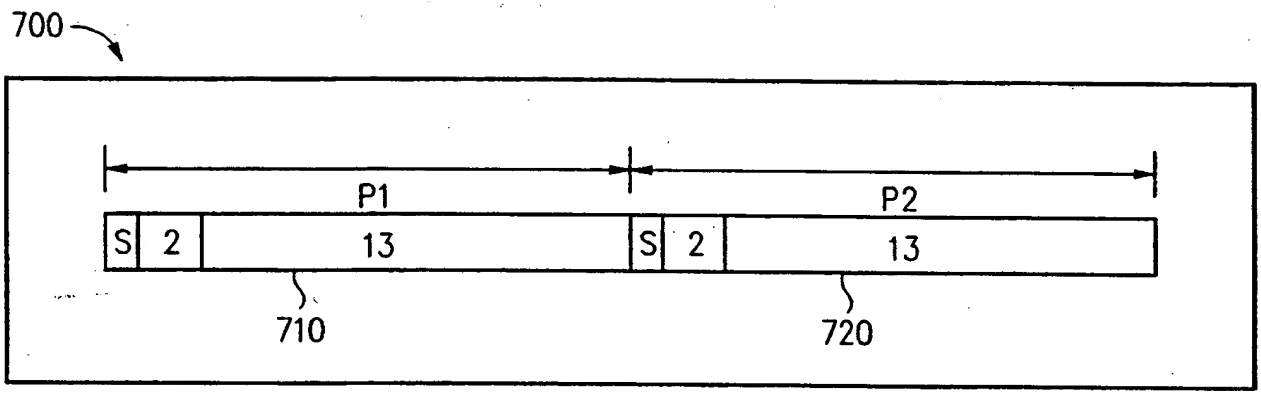


FIG. 7

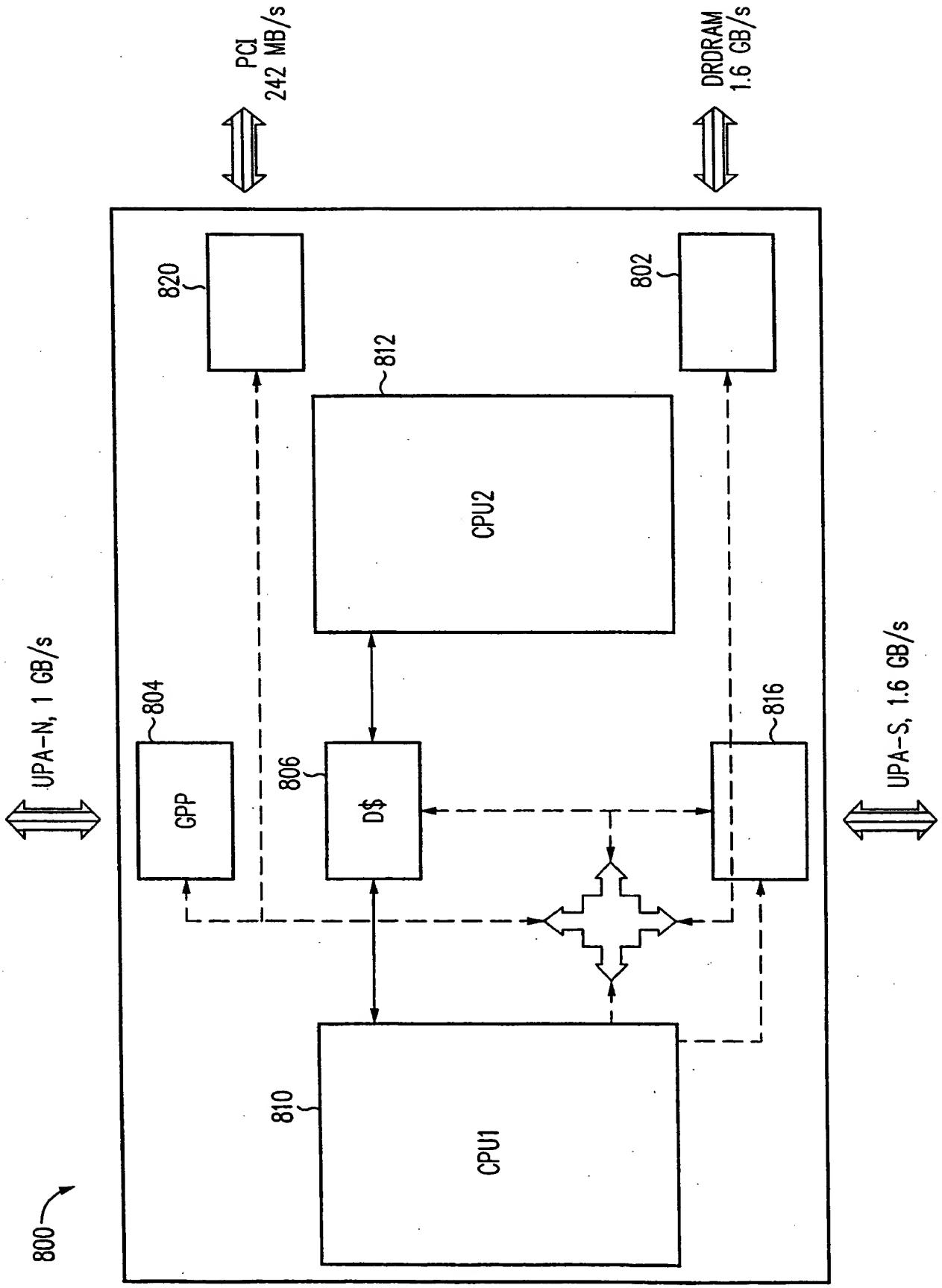


FIG. 8

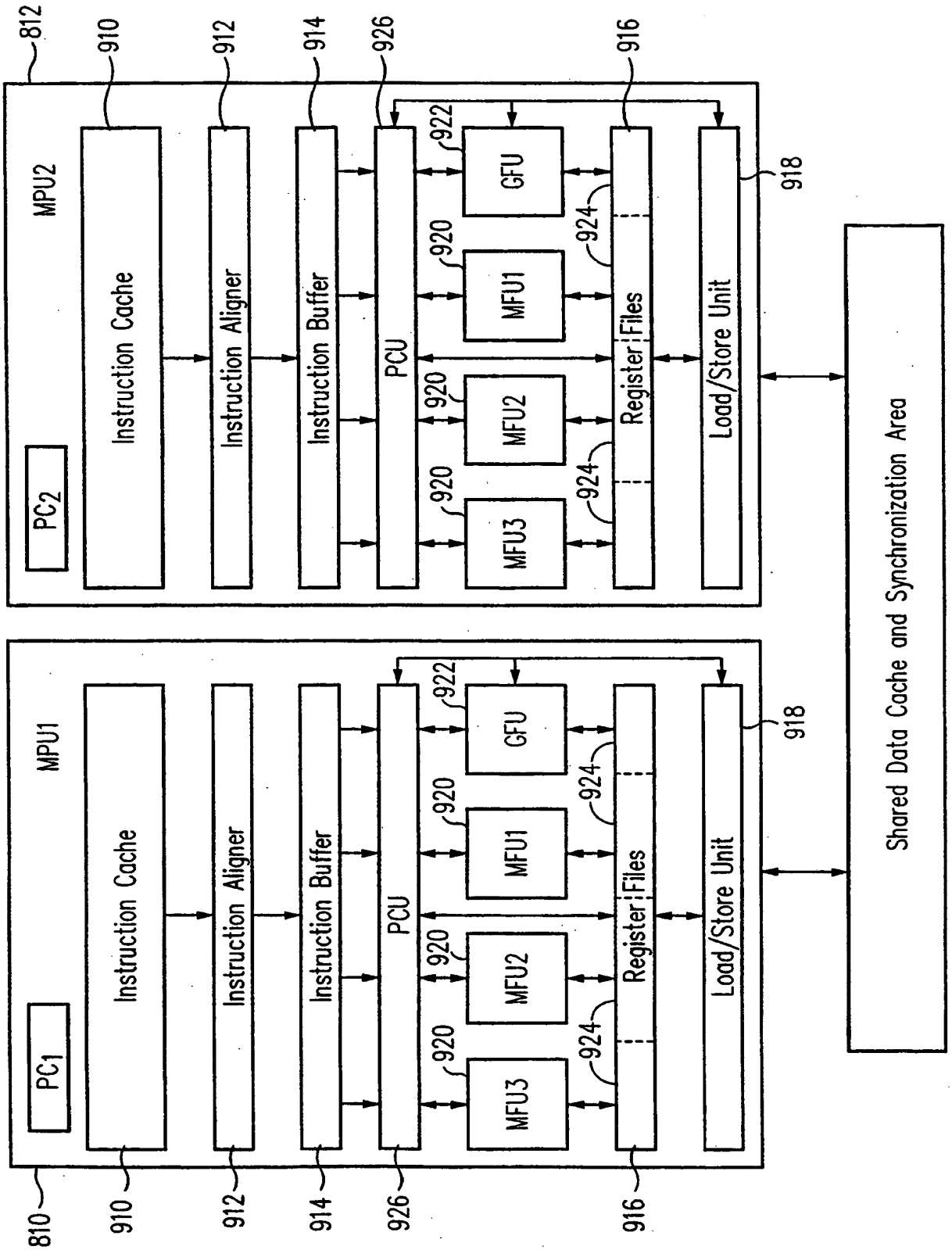


FIG. 9

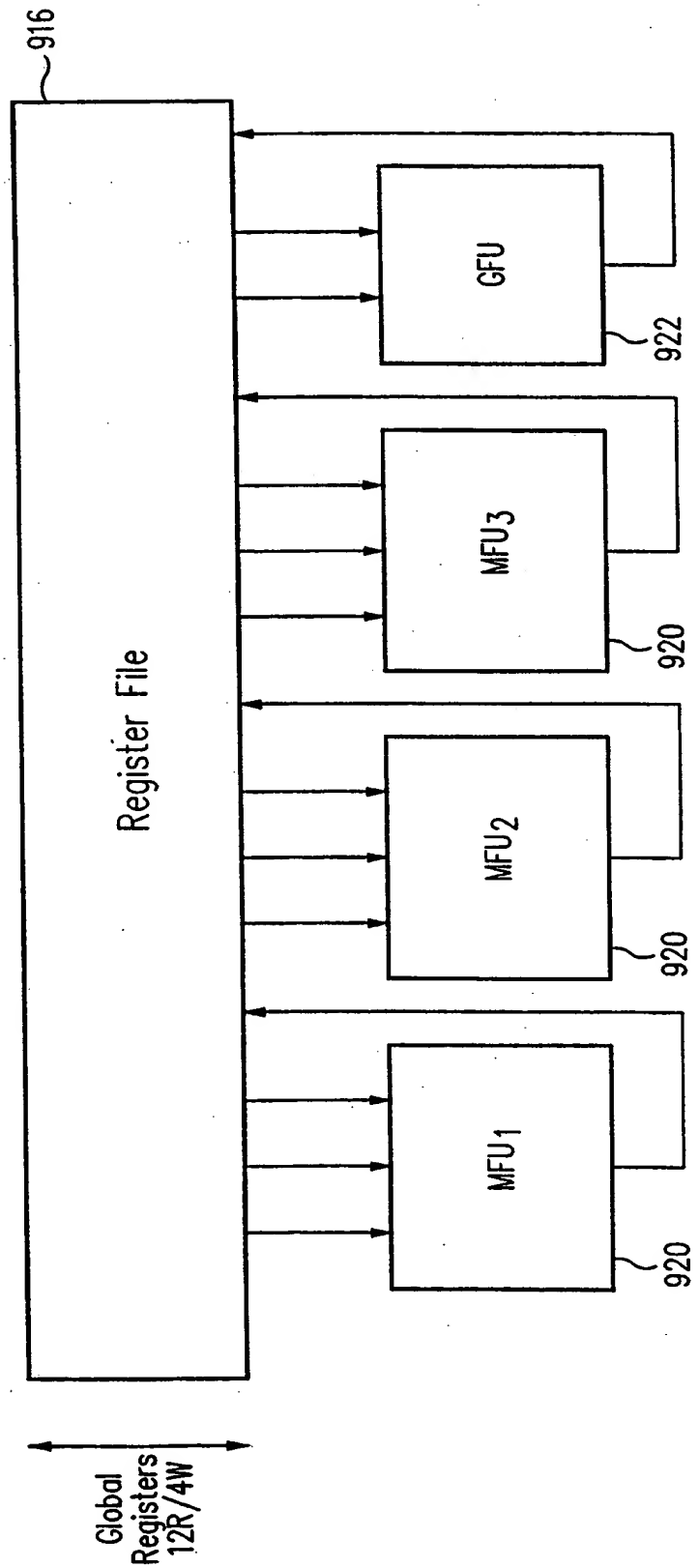


FIG. 10

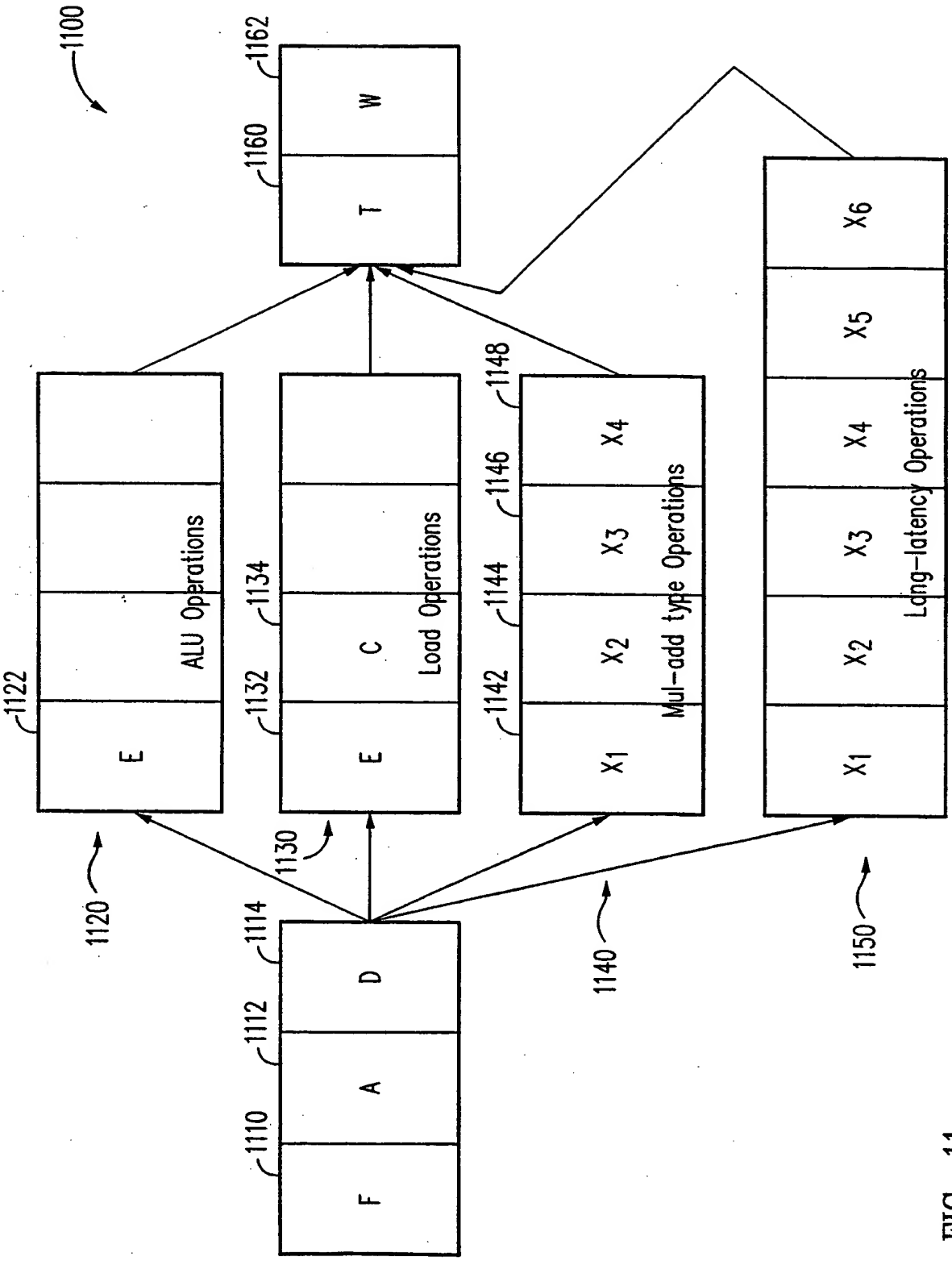


FIG. 11

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 August 2000 (03.08.2000)

PCT

(10) International Publication Number
WO 00/45251 A3

(51) International Patent Classification[7]: G06F 7/552

(21) International Application Number: PCT/US00/01578

(22) International Filing Date: 24 January 2000 (24.01.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/240,977 29 January 1999 (29.01.1999) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventors: SHANKAR, Ravi; 1360 Los Arboles Avenue, Sunnyvale, CA 94087 (US). SUDHARSANAN, Subramania, I.; 4340 Cambridge Way, Union City, CA 94587 (US).

(74) Agents: KOESTNER, Ken, J. et al.; Skjerven, Morrill, MacPherson, Franklin & Friel LLP, Suite 700, 25 Metro Drive, San Jose, CA 95110 (US).

(81) Designated State *(national)*: JP.

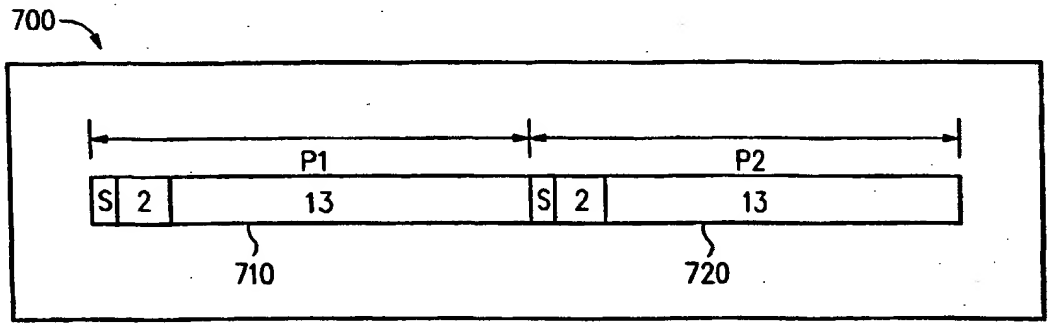
(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published:
— *With international search report.*

(88) Date of publication of the international search report:
14 December 2000

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: FLOATING AND PARALLEL FIXED POINT SQUARE ROOT AND RECIPROCAL POINT SQUARE COMPUTATION UNIT IN A PROCESSOR



(57) Abstract: A parallel fixed-point square root and reciprocal square root computation uses the same coefficient tables as the floating point square root and reciprocal square root computation by converting the fixed-point numbers into a floating-point structure with a leading implicit 1. The value of a number X is stored as two fixed-point numbers. In one embodiment, the fixed-point numbers are converted to the special floating-point structure using a leading zero detector and a shifter. Following the square root computation or the reciprocal square root computation, the floating point result is shifted back into the two-entry fixed-point format. The shift count is determined by the number of leaded zeros detected during the conversion from fixed-point to floating-point format.

WO 00/45251 A3

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/01578

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F7/552

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 245 564 A (HU LARRY ET AL) 14 September 1993 (1993-09-14) abstract column 3, line 46 - line 52	1-14
A	PATENT ABSTRACTS OF JAPAN vol. 15, no. 455 (P-1277), 19 November 1999 (1999-11-19) & JP 03 192429 A (FUJITSU), 22 August 1991 (1991-08-22) abstract; figure 6	1-14

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 June 2000	14 07 2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Cohen, B

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/01578

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	"A Table Based Method to Compute Square Root" RESEARCH DISCLOSURE., vol. 41, no. 408, April 1998 (1998-04), XP002140018 INDUSTRIAL OPPORTUNITIES LTD. HAVANT., GB ISSN: 0374-4353 Paragraph 2 ('The algorithm, briefly, is as follows:...'), line 8-11, 19-21.	1-14
A	EP 0 132 646 A (IBM) 13 February 1985 (1985-02-13) abstract page 5, line 13 -page 6, line 14 page 15, line 21 -page 16, line 16	1-14
A	SU 1 381 494 A (TAGANROGSKIJ RADIOTECH INST) 15 March 1988 (1988-03-15) abstract	1-14
A	EP 0 380 100 A (HUGHES AIRCRAFT CO) 1 August 1990 (1990-08-01) abstract	1-14

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

ernational application No.
PCT/US 00/01578

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☒ Claims Nos.: 1-14 (in part)
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:
see FURTHER INFORMATION sheet PCT/ISA/210

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

Continuation of Box I.2

Claims Nos.: 1-14 (in part)

Present claims 1-14 relate in varying amounts of detail to a method/apparatus defined by reference to a desirable characteristic or property, namely approximating the square root of two fixed point numbers by using a floating point computing device.

The claims cover all methods/apparatuses having this characteristic, whereas the application provides support within the meaning of Article 6 PCT and/or disclosure within the meaning of Article 5 PCT for only a very limited number of such methods/apparatus. In the present case, the claims so lack support, and the application so lacks disclosure, that a meaningful search over the whole of the claimed scope is impossible. Independent of the above reasoning, the claims also lack clarity (Article 6 PCT). An attempt is made to define the method/apparatus by reference to a result to be achieved. Again, this lack of clarity in the present case is such as to render a meaningful search over the whole of the claimed scope impossible. Consequently, the search has been carried out for those parts of the claims which appear to be clear, supported and disclosed, namely those parts relating to the methods/apparatus mentioned in the description at page 11, line 14 - page 13, line 8.

The applicant's attention is drawn to the fact that claims, or parts of claims, relating to inventions in respect of which no international search report has been established need not be the subject of an international preliminary examination (Rule 66.1(e) PCT). The applicant is advised that the EPO policy when acting as an International Preliminary Examining Authority is normally not to carry out a preliminary examination on matter which has not been searched. This is the case irrespective of whether or not the claims are amended following receipt of the search report or during any Chapter II procedure.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 00/01578

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5245564 A	14-09-1993	US 5280439 A	18-01-1994
JP 03192429 A	22-08-1991	NONE	
EP 0132646 A	13-02-1985	US 4594679 A	10-06-1986
		AR 241060 A	30-04-1991
		BR 8401796 A	19-03-1985
		DE 3483929 D	21-02-1991
		JP 1730560 C	29-01-1993
		JP 4014366 B	12-03-1992
		JP 60027026 A	12-02-1985
SU 1381494 A	15-03-1988	NONE	
EP 0380100 A	01-08-1990	US 4953119 A	28-08-1990
		AU 646339 B	17-02-1994
		AU 1805692 A	30-07-1992
		AU 4884090 A	16-08-1990
		CA 2007054 A,C	27-07-1990
		JP 1998844 C	08-12-1995
		JP 2240728 A	25-09-1990
		JP 7027457 B	29-03-1995